

المرجع المبسط في

البرمجة كائنية التوجه

Object-Oriented Programming

تأليف وإعداد

أحمد نجم

المرجع المبسّط في

البرمجة كائنية التوجه

يضم الكتاب المواضيع التالية :

- الفئات Class.
- التراكيب Structure.
- مبادئ البرمجة كائنية التوجه OOP Concepts.
- وفيما بين السطور بعض الأمثلة وبعض النقاط المبهمة للكثير من المطورين.
- بالتوفيق للجميع بإذن اللّه تعالى ، وانتظروا المرجع الشامل للبرمجة كائنية التوجه في القريب إن شاء اللّه.



فهرس

مُقَدِّمَة

بسم الله الذي لا يضر مع اسمه شيء في الأرض ولا في السماء وهو السميع العليم

بسم الله الرحمن الرحيم

❁❁❁❁❁❁❁❁❁❁❁❁❁❁❁

في البداية أشكر من لا يُشكر ولا يُحمد على مكروهٍ سواه ﷻ أن وفقني جل في علاه لما أنا عليه الآن ، فالحمد لله حتى يرضى والحمد له كل الحمد بعد الرضا. الحمد لله أن وفقني لإعداد هذا الكتاب المتواضع.

بداية أشكر كذلك إخواني في الله ، والذين قد ساعدوني في تنقيح هذا الكتاب فنيا حتى تصح المعلومات والأمثلة الواردة فيه :

- م / أحمد جمال خليفة.
- أ / محمد أسعد.
- أ / محمد أسامة جندية.

ولحسن حظك أخي القارئ أن من قاموا بالتنقيح وتصحيح الكتاب هم أساتذة لي وعلى قدر من العلم يُحسدون عليه - زادهم الله علما ووفقهم الله لما يحبه ويرضاه - ، ولقد وفقني الله تعالى إلى الحديث عن البرمجة كائنية التوجه أو ما يسمى بالـ **Object-Oriented Programming** ، وكذلك الفئات والتراكيب وكان بودي لو أنني أستطرد وأزيد تفصيلا في الحديث في هذا الصدد وكذلك كان بودي أن

أتحدث بالتفصيل عن الـ Namespaces وكذلك الـ Generics و الـ Collection Classes و ...الخ ، ولكن كان الوقت عاملا في غير صالحي ، ولكن بحمد الله فإني أعد الكتاب كمرجع مبسط للدارسين ولمن لا يعرفون الكثير عن البرمجة كائنية التوجه والتي تمثل المنطق الخاص بالمبرمج أثناء تطويره للتطبيقات ولا غنى لأي مبرمج عن مبادئ ومفاهيم الـ OOP ، وبإذن الله تعالى سيتم إصدار مرجع موسع عما قريب بإذن الله جل في علاه حين أنتهي من مشاغلي ... ولكن هناك بعض النقاط التي يجب أن نذكرها قبل البدء :

- ستكون الأمثلة المتاحة في هذا الكتاب مطورة تحت بيئة Visual Studio .NET 2008.
- سنستخدم لغة Visual Basic .NET 2008 أثناء تطوير الأمثلة.
- آمل أن يكون لدى القارئ خلفية أو دراية مسبقة بلغة Visual Basic .NET أو بأي لغة أخرى تدعم البرمجة كائنية التوجه ، وكذلك تقنية NET. بشكل عام.
- سعر الكتاب .:: أن تدعو للمسلمين كافة بالرحمة والمغفرة ونيل رضا الله – جل في علاه – وكذلك شفاعة النبي المصطفى صلى الله عليه وسلم.
- قد استعنت في إعدادي لهذا الكتاب البسيط بكتاب Visual Basic® .NET 2008 لكاتبه Rod Stephens ، وقد قمت بترجمة الكثير منه وصياغته بأسلوب مبسط وكذلك اقتباس بعض الأكواد والأمثلة التوضيحية.

بالتوفيق للجميع ، وشكرا لإخواني الذين ساعدوني في إخراج الكتاب بهذه الصورة

مع تحياتي ... المؤلف

تعريف بالمؤلف

❁ أحمد محمد عبد العظيم نجم (أحمد نجم)

❁ جمهورية مصر العربية / محافظة المنيا

❁ البريد الإلكتروني : AhmedNegm@WindowsLive.Com

❁ الموبايل : 002 012 944 79 49 or 002 011 977 72 44

❁ مكتبة المواضيع المميزة :

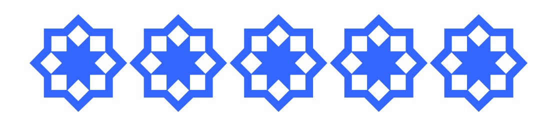
تحتوي مكتبة مواضيعي المميزة على بعض الدروس والمقالات التي قمت بنشرها على الشبكة في منتديات أكاديمية فيجوال بيسك للعرب وكذلك منتديات فيجوال بيسك للعرب.

http://www.vba4a.com/vb/showthread.php?t=429

أسأل الله تعالى أن يجعل هذا عملا صالحا وأن يكون لوجهه تعالى خالصا ولا يشرك فيه أحد سواه جل في علاه ، وأن يكون هذا العمل ذخرا لي ولإخواني يوم نلقى الله تعالى ، وأسأل الله أن يغفر لنا ذنوبنا وإسرافنا في أمرنا في هذه الأيام المباركة.
آمين

أولا : الفئات والتراكيب

Classes and Structures



دعونا نبدأ سريعا بدون مقدمات ... فهنا نحن بصدد الحديث عن الفئات والتراكيب ، ومن ثم نبدأ أولا بالحديث عن (المتغير Variable) وهو عادة يحمل قيمة مفردة وقد تكون هذه القيمة عددية Number أو نصية String.. الخ ، أو قد يكون كمرجع أو كمؤشر لنقطة تشير إلى كائن عام أو صورة ... الخ.

سنتطرق سويا إلى ماهية الفئات والتراكيب وكذلك كيفية التصريح عن كل منهما وكيف ننشأ نسخة Instance منهما أيضا ، كما سنوضح كذلك الفرق بين الفئة Class والتركيب Structure ، وسنحاول جاهدين تقديم بعض النصائح الخاصة بوقت أو حالة استخدام كل منهما .. تابع معي.



الفئات Classes

كي نتعامل مع الفئات فيجب أن يكون لكل فئة أحداثها Events وخصائصها Properties ووسائلها (طرقها) Methods. فلو تناولنا عناصر الفئة الثلاث المذكورين فيما سبق فسيكون الآتي :

- الخاصية (Property) ... الخاصية هي إحدى الطرق التي عن طريقها يتم تخزين نوع أو شكل معين من أنواع البيانات ، ومن الممكن أن تكون هذه القيمة بسيطة كاسم أو رقم أو تاريخ ، ومن الممكن أن تكون معقدة نسبيا كالمصفوفات Arrays أو تكون الخاصية أساسا عبارة عن

كائن مستقل له خصائصه وأحداثه وطرقه أو وظائفه. للعلم هناك خصائص من النوع (Read/Write) وهناك نوع آخر وهو (Read-Only) ، وكذلك النوع (Write-Only) ومثال ذلك خصائص كتابة كلمات المرور Passwords.

- **الطريقة أو الوسيلة (Method) ...** تمثل الطريقة دالة أو إجراء (Function Or Subroutine). وتعتبر الطريقة أو الوظيفة كجزء من الكود داخل الفئة Class مما يضفي عليها السمة الأساسية للفئات وكذلك تواجد الطرق داخل الفئات لتنفيذ مجموعة من سطور الكود ضمنيا وتنفيذ وظيفة أو مهمة معينة ظاهريا.
- **الحدث (Event) ...** الحدث بكل بساطة هو عبارة عن إخطار حركي. يعني هذا أن الحدث يستدعي قِطع أو إجراءات معينة من الكود داخل الـ Class كي يخبرها أن بعض الأحداث أو الشروط داخل الفئة قد وقعت بالفعل.

أعلم بالطبع أني لم أضف لديك جديدا بل بكل تأكيد قد شوشت عقلك وذهبت ببعض المفاهيم لديك إلى حيث لا تدري ، ولكن أصغ لي مرة أخيرة وتخيل معي هذا المثال البسيط. بفرض مثلا أنه لدينا فئة أو Class بالمسمى (Job) ، فهيا بنا نحلل ما جاء بها كما يلي :

أولا ـ الخواص (Properties) :

- JobDescription (وصف أو تعريف الوظيفة) = وهي خاصية تحتوي على وصف نصي للوظيفة.
- WorkHours (ساعات العمل) = وهي خاصية تحتوي على عدد ساعات العمل لتلك الوظيفة.
- JobCustomer (المتقدم للوظيفة) = وهنا ستختلف نوع الخاصية فهنا سنقوم بتعريف الخاصية JobCustomer على أنها عبارة عن كائن مشتق من فئة Class آخر بالاسم Customer. والفئة Customer تحتوي على خصائص وأحداث وطرق ووظائف ومتغيرات وكذلك دوال وإجراءات.

- EstimatedHours (الساعات المقدرة) = عدد الساعات المبدئية المقدرة لأداء هذه الوظيفة.

ثانيا ـ الطرق أو الوسائل (Methods) :

- AssignJob (اعتماد الوظيفة) = هذا الإجراء يقوم بتنفيذ اعتماد الوظيفة من الموظف المختص بالتعيين إلى العميل المتقدم للوظيفة.
- BillJob (كشف حساب) = يقوم مثلا هذا الإجراء بطباعة كشف حساب على سبيل المثال للشخص المتقدم للوظيفة.
- EstimatedCost (التكلفة المقدرة) = دالة تقوم بالرجوع بالتكلفة المقدرة بناءا على ما ورد في عقد خدمات العميل أو المتقدم للوظيفة وكذلك الساعات المقدرة التي وردت بالخصائص سابقا EstimatedHours.

ثالثا ـ الأحداث (Events) :

- Created (إنشاء الوظيفة) = يوفر هذا الحدث إمكانية التحكم بالفئة أو التطبيق عند إنشاء الوظيفة عن طريق كتابة ما يشاء في هذا الحدث.
- Assigned (التعيين) = يتم تنفيذ ما ورد في هذا الحدث عند موافقة الموظف على العميل المتقدم للوظيفة وإعطاء أمر التعيين.
- Rejected (الرفض أو ترك المهمة) = يتم تنفيذ هذا الحدث عند رفض إكمال المهمة من قِبَل الموظف ، ربما حدث هذا لعدم اكتمال أوراق العميل أو ربما لظروف طارئة للموظف أو عدم تواجد الخبرة الكافية للقيام بهذه المهمة.
- Canceled (إلغاء الأمر) = وهذا مثلا يتم من قِبَل العميل قبل إصدار أمر التعيين.

كانت هذه نظرة سريعة على فئة أو Class تحتوي على أحداث وخصائص ودوال وإجراءات وهذا فقط على سبيل المثال. نأتي بعد ذلك لملحوظة مهمة وهي مجمل ما سبق فتعريف الفئة هي عبارة عن مصدر أو كيان أو شخصية برمجية مستقلة تغلف أو تحتوي على مجموعة

من التعليمات البرمجية على اختلاف أنواعها (خصائص أو أحداث أو دوال وإجراءات). ولكن كن على علم بأنه ليس من الضروري وجود كل المكونات الثلاثة السابقة داخل كل فئة ، ولكن نحن هنا بصدد الحديث عن الأحوال القياسية لمكونات الفئات Classes.

انظر معي إلى الصيغة القياسية لكتابة الفئات :

```
[attribute_list] [partial] [accessibility] [shadows] [inheritance] _
Class name[(Of type_list)]
     [Inherits parent_class]
     [Implements interface]
     Statements
End Class
```

** ملحوظة : العبارات بين الأقواس "[" وكذلك "]" هي اختيارية الكتابة **

الشيء الوحيد الذي يجب كتابته أو تواجده عند التصريح عن الفئات هو كلمة "Class" التي تسبق اسم الفئة والمحدد باللون الأزرق في الكود السابق في السطر الثاني ، وكذلك أيضا العبارة "End Class" المحدد كذلك باللون الأزرق في السطر الأخير من الكود السابق. كل شيء عدا ما سبق يعتبر اختياري ، حتى إنك على سبيل المثال يمكنك التصريح عن فئة بدون اللجوء للاختيارات السابقة كما يلي :

```
Class clsPerson
     Statements
End Class
```

حيث clsPerson هو اسم الفئة المصرح عنها. ولو تطرقنا سريعا للتصريحات أو التعبيرات الاختيارية أثناء التصريح عن الفئات فسنقول ما يلي :

- **Attribute_List** :

هي عبارة عن قائمة خاصة من الصفات التي تمنح للفئات. واستخدامك لـ attribute يعطي سمة خاصة للفئة وذلك لإعطاء الـ Compiler معلومات أكثر تفصيلا عن الفئة وكذلك وقت التشغيل. هذه الصفات أو السمات أو ما يسمى بالـ attribute متخصصة نوعا ما ، فهب مثلا أنك تصنع أحد التطبيقات ولابد أن يدعم هذا التطبيق السحب والإفلات drag-and-drop وذلك لأخذ نسخة من فئة في تطبيق إلى تطبيق آخر ، فهنا يجب عليك وصف الفئة على أنها serializable كما هو موضح كما يلي :

```
<Serializable()> _
Class clsPerson

    Public FirstName As String
    Public LastName As String

End Class
```

لن أتحدث أكثر من ذلك في هذه النقطة ، لأني من الأساس لا أعيها كاملا ، ولكن سأتركك مع بعض الروابط المفيدة في هذا الصدد :

http://msdn.microsoft.com/en-us/library/system.serializableattribute.aspx

- **Partial** :

العبارة أو الكلمة الدليلية partial تقوم بإعلام الـ Visual Basic .NET بأن التصريح التالي عبارة عن تعريف جزء من الفئة ، في الكود التالي ستجد بأن الفئة clsPerson تم كسرها أو تعريفها أو التصريح عنها على جزأين كما يلي :

```
Partial Class clsPerson

    Public FirstName As String
    Public LastName As String

End Class
''''''''''''''''''''''''''''''''''''''
Partial Class clsPerson

    Public Age As Integer
    Public EMail As String

End Class
```

التطبيق أو البرنامج يمكن أن يحوي بداخله أكثر من قطعة كود أو أكثر من تعريف للفئة clsPerson ، وكذلك يمكن وجود قطع الكود هذه في أكثر من وحدة نمطية Module ، ويقوم الـ Visual Basic .NET بتجميع أو دمج هذه القطع سويا لتكوين الفئة التي نريد.

تتساءل بالطبع : ما الفائدة من هذا ؟؟ ... واحدة من الفوائد الأساسية هي تجميع وحدات الكود المرتبطة مع هذه الفئة أو المتعلقة بها في حزمة مفردة. ولكن نصيحة لا تقم بتقسيم الفئة ما دمت لا تملك سببا وجيها لتقسيمها ، فهب مثلا لو أنك سمحت لعدد من المطورين بالعمل على وحدات

مجزئة من الفئة كل على حده في نفس التوقيت ، أو بعض من الوحدات الأخرى لا تريد لأحد الاطلاع عليها.

يجب تواجد الكلمة الدليلية Partial في قطعة واحدة من الفئة المجزأة على الأقل ، وتواجد الكلمة في القطع الأخرى اختياري أي أن عدم وجوده لن يؤثر على تجميع الفئة. إلى هنا يكون قد اتضح أن استخدام الكلمة الدليلية في جميع القطع المجزأة من الفئة يؤكد بأن تجزئة الفئة أو كسرها أو التصريح عنها في أكثر من موضع يقلل من التعارض والتشويش أثناء التطوير فيها.

- **Accessibility** :

نتحدث في هذه النقطة عن قابلية الوصول أو ما يسمى بالـ Accessibility ، وهنا أنت بصدد استخدام معرف أو كلمة دليلية من الآتي :

- **Public** :

تشير هذه الكلمة المفتاحية إلى قابلية الوصول للفئة من خلال الكود سواء من داخل الفئة ذاتها أو خارجها. فهذا يمكن الغالبية العظمى من الوصول للفئة وكذلك أي كود يستطيع التعامل مع النسخة المأخوذة من الفئة. اعلم انه يمكنك فقط استخدام الكلمة المفتاحية Protected إذا ما احتوت الفئة على فئة أخرى بداخلها. تابع الكود التالي :

```
Public Class clsPerson

    Public FirstName As String
    Public LastName As String
    Protected Address As PersonAddress

    Protected Class PersonAddress
        Public Street As String
        Public City As String
    End Class

End Class
```

ولأن الفئة PersonAddress تم التصريح عنها بالكلمة المفتاحية Protected فهي مرئية بالفعل داخل نطاق الفئة clsPerson وكذلك أي فئة مشتقة من هذه الفئة ، ولا تقلق من كلمة "مشتقة" فسنتحدث لاحقا عن البرمجة كائنية التوجه وتعريف وراثة الكائنات.

- **Friend** :

تشير هذه الكلمة المفتاحية إلى قابلية الوصول للفئة من خلال الكود سواء داخل الفئة أو خارجها ولكن فقط داخل المشروع الحالي. الفرق بين Friend و Public أن الثانية تسمح للكود بالوصول لمكونات الفئة من خارج المشروع نهائيا أي من داخل الملفات من النوع (DLL). وعلى سبيل المثال فلنفرض جدلا أنك تقوم بكتابة تطبيق من النوع Control Library ويحوي داخله العديد من الإجراءات والدوال ... الخ ، فإذا تم تعريف الفئة أو التصريح عنها باستخدام الكلمة المفتاحية Public فالكود المكتوب داخل المكتبة (dll.) والكود المكتوب داخل المشروع يستطيع الوصول لما هو تحويه هذه الفئة من وسائل ... الخ. أما لو تم التصريح عن الفئة باستخدامك للكلمة المفتاحية Friend فلن تستطيع الوصول لأي مما كُتِب داخل الفئة من كود إلا عن طريق المكتبة ذاتها (dll.) ولا تستطيع ذلك إذا قمت بإضافة المكتبة كمرجع في تطبيق آخر.

- **Protected Friend** :

تعتبر الكلمة المفتاحية Protected Friend كمزيج أو اتحاد من الكلمتين Protected و Friend ، فالفئة المصرح عنها باستخدام الكلمة المفتاحية Protected Friend قابلة للوصول إليها من خلال فئة أخرى في نفس المشروع وكذلك الفئات المشتقة ولكن داخل نفس المشروع.

- **Private** :

تشير هذه الكلمة المفتاحية إلى قابلية الوصول للفئة من خلال الكود ولكن داخل نفس وحدة الكود ، هذا يعني لو أنه كان لديك متغير مصرح داخل فئة باستخدام الكلمة المفتاحية Private فهي معرفة داخل الفئة فقط وقس على ذلك إذا كان المتغير مصرح داخل الـ Module أو داخل Structure ... الخ.



إذا لم تستخدم أية كلمات أو عبارات مما سبق ذكرها فإن الكلمة الافتراضية التي يتم إسنادها للمتغير أو الـ Method ... الخ هي الكلمة المفتاحية Friend.

- **Shadows** :

الكلمة الدليلية Shadows تؤدي إلى إخفاء تصريحات بعض من الكيانات المصرح عنها داخل فئة أخرى وهي الفئة الأب باعتبار أن الفئة الحالية هي الفئة المشتقة.

في الكود التالي لدينا الفئة Employee والتي بداخلها فئة أخرى بالاسم OfficeInfo ولدينا كذلك كائن بالاسم Office وهو عبارة عن نسخة من الفئة OfficeInfo. وفرضا كذلك أنه لدينا فئة أخرى بالاسم Manager وهي فئة مشتقة من الفئة السابقة ، وفي الفئة Manager سنقوم بتعريف شكل جديد للفئة الفرعية OfficeInfo باستخدام الكلمة الدليلية Shadows ، والكائن الذي ينسخ من الفئة OfficeInfo بالاسم ManagerOffice.

```
'First Class
Public Class Employee

    Public Class OfficeInfo
        Public OfficeNumber As String
        Public Department As String
    End Class

    Public FirstName As String
    Public LastName As String
    Public Office As New OfficeInfo

End Class

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Second Class
Public Class Manager

    Inherits Employee

    Public Shadows Class OfficeInfo
        Public OfficeNumber As String
        Public Department As String
        Public SecartaryOfficeNumber As String
        Public SecrtaryDepartment As String
```

```
    End Class

    Public ManagerOffice As New OfficeInfo

End Class
```

سأختصر عليك قليلا بقولي لك : تابع القسم الخاص بـ Shadows لاحقا حيث نتحدث عن OOP ، ولكن هنا حتى ننتهي من هذه النقطة سنقول ما يلي ... في الكود التالي سنقوم باستخدام الفئتين Employee و Manager. سنقوم بكتابة الآتي :

```
Dim emp As New Employee
Dim mgr As New Manager

emp.Office.Department = "KU 600"
mgr.Office.Department = "MX 775"
mgr.ManagerOffice.SecrtaryDepartment = "WQ 564"
```

فضلا ، راجع القسم الخاص بـ Shadows لاحقا في OOP في الجزء الآخر من هذا الكتاب ، لأن المجال الآن لا يسع للحديث عنه.

- **Inheritance** :

الوراثة ... لن نتحدث كثيرا عن الوراثة أو الاشتقاق ، ولكن من اسمه تستطيع أن تفهمه مؤقتا ، المهم هنا أن الفئة الموروثة من فئة أخرى ترث كل خصائص وأحداث وعناصر هذه الفئة ، وتسمى الفئة الأصل (الفئة الأب Parent Class) .. أما الفئة المشتقة أو الوارثة تسمى بالابن Derived Class ، وسنتحدث لاحقا بكل تفصيل عن الوراثة عندما نصل للحديث عن البرمجة الكائنية التوجه في منتصف الكتاب بإذن الله.

- **Of type_list** :

من العبارات الجيدة التي قرأتها في صدد (قائمة النوع Type List) كان ذلك :

The *Of type_list* clause makes class generic.

فهي فعلا تعطي قابلية للتطبيق بإنشاء كائن من الفئة والذي يتعامل مع نوع محدد من البيانات. فعلى سبيل المثال فإن الكود التالي به تصريح عن فئة بالاسم Tree محددة النوع ، والفئة تحتوي على متغير بالاسم RootObject وهو من نفس نوع البيانات المعطى للفئة.

```
Public Class Tree(Of data_type)
```

```
    Public RootObject As data_type
End Class
```

لو قرأت التصريح السابق ستعتقد بأن الفئة Tree هي متفرعة فعلا من شيء ما. إذا نظرنا للمثال التالي حيث المتغير my_tree بوضعه كتفريع من الفئة Employee مثلا :

```
Dim my_tree As Tree(Of Employee)
my_tree = New Tree(Of Employee)
my_tree.RootObject = New Employee
```

أرجو المعذرة في إبهامي لهذه النقطة ، ولكن أخشى الخوض فيها ولكن يمكنك البحث في MSDN حول هذه الكلمة " Generics " ، وقم بالاضطلاع على سمات Generic Classes بنفسك إلى أن يوفقني الله تعالى بشرحها أو الحديث عنها. أو مؤقتا اضطلع على الرابط التالي :

<http://msdn.microsoft.com/en-us/library/w256ka79(VS.80).aspx>

❁❁ الفرق بين الكائن والفئة ❁❁

بعد الحصول على نسخة من الفئة ووضع محتوياتها داخل كيان آخر عن طريق جمل التصاريح مهما اختلفت صيغتها كمثل :

```
Dim MyObj As New Class1
```

فهنا يتضح الفرق بين الكائن والفئة على النحو التالي ، فهنا الفئة هي المسماة Class1 ، أما الكائن فالمقصود به الكيان البرمجي الذي قمنا بوضع كافة خصائص الفئة بداخله وهو المسمى في مثالنا MyObj.

❁❁ ملحوظة مهمة ❁❁

يعتبر الـ Visual Basic .NET مكتظ بالفئات ، فعلى سبيل المثال تعتبر كل أداة موجودة داخل الـ Component عبارة عن Class مستقلة لها أحداثها وخصائصها وطرقها ، وتعتبر الفئة الأب هي الفئة المسماة Control. وعند استخدامك لهذه الأدوات فما أنت تعمل إلا مع كائنات مشتقة من مصدرها الأساسي ، فلو فرضنا أننا لدينا TextBox على الـ Form فمربع النص هذا عبارة عن كائن مشتق من الفئة المسماة TextBox ويحمل هذا الكائن كافة ما

يميز هذه الفئة المشتق منها. باختصار وكما يقول البعض (Visual Basic .NET is jam-packed with classes).

❁❁ التراكيب Structures ❁❁

يتشابه التركيب الكودي Structure كثيرا مع الفئات ، فلو نظرنا سريعا إلى النص القياسي للتصريح عن تركيب ما فسيكون كما يلي :

```
[attribute_list] [partial] [accessibility] [shadows] _
Structure name[(Of type_list)]
     [Implements interface]
     Statements
End Structure
```

** ملحوظة : العبارات بين الأقواس [وكذلك] هي اختيارية الكتابة **

الشيء الوحيد الذي يجب كتابته أو تواجده عند التصريح عن التراكيب هو كلمة " Structure " التي تسبق اسم التركيب والمحدد باللون الأزرق في الكود السابق في السطر الثاني ، وكذلك أيضا العبارة " End Structure " المحدد كذلك باللون الأزرق في السطر الأخير من الكود السابق. كل شيء عدا ما سبق يعتبر اختياري ، حتى إنك على سبيل المثال يمكنك التصريح عن تركيب ما بدون اللجوء للاختيارات السابقة كما يلي :

```
Structure EmptyStructure
     Private MyNumber As Integer
End Structure
```

ولكن هنا أنت لست بصدد التعامل مع الفئات ، فهنا يجب ألا يكون التركيب فارغ من الأكواد بعكس الحال مع الفئات ، ولكن يجب على الأقل أن يحتوي التركيب على متغير واحد أو ... الخ.

ويجب أن تعلم تماما أن العبارات الاختيارية وهي [attribute_list] و [partial] و [accessibility] و[shadows] وكذلك جملة تنفيذ أو تطبيق الواجهات Implements فهي كما هي في الفئات كما تناولنها سابقا.

❁❁ من الفروق الواضحة بين الفئات والتراكيب هي :

- <u>لا تستطيع الوراثة داخل التركيب</u> :

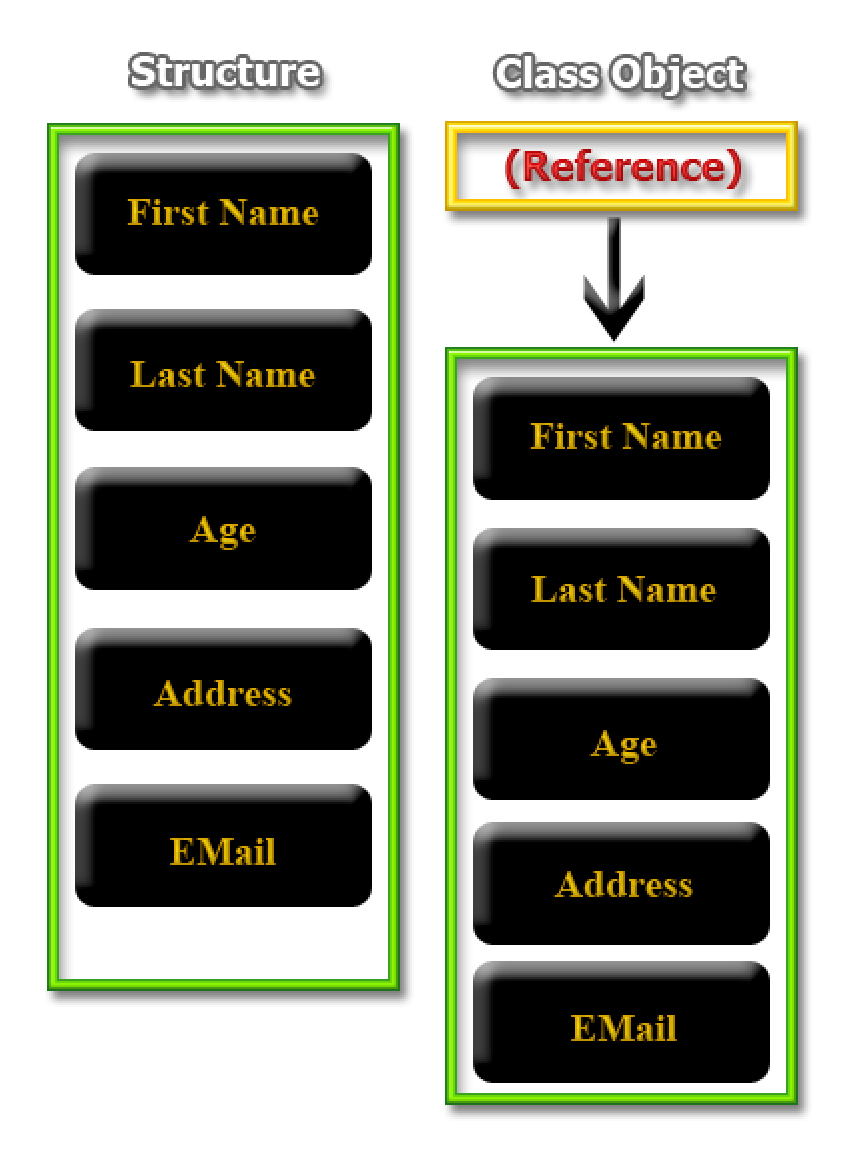
نعم ، لا يمكنك الوراثة أو الاشتقاق من فئة أخرى بعكس التعامل مع الفئات ولذلك هنا لا يمكنك استخدام الكلمات الدليلية MustInherit ولا NotInheritable أو حتى الكلمة المفتاحية Inherits ولكن يمكنك تنفيذ أو تطبيق أي واجهة Interface داخل

التركيب Structure. وبالنسبة للحديث عن الواجهات فهذا سيتم مناقشته تفصيلا فيما يخص الـ OOP لاحقا.

- **التركيب من النوع ذات القيمة Value Type :**

من الفروق الواضحة جدا بين الفئة والتركيب هو كيفية تخزين متغيراتهما وملحقاتهما داخل الذاكرة. الفئات من النوع ذات المرجع (Reference Types) وهذا يعني أن نسخة ما من الفئة ما هي إلا كمؤشر يشير إلى كائن في الذاكرة RAM. أي أنك إذا أخذت نسخة من فئة ما فإن Visual Basic يقوم بإنشاء مؤشرات أو مراجع تشير إلى كائن ما يسكن أو يقطن فعليا في الذاكرة.

على الجانب الآخر .. يعتبر التركيب من النوع ذات القيمة Value Type. النسخة المأخوذة من التركيب Instance تتكون من البيانات أو الحقول الموجودة بداخل التركيب بدلا من الإشارة إليه. انظر للشكل التالي فهي توضح شيئا مما سبق :





المشيدات والمهدمات

سنتعرض سريعا الآن لما يسمى (المشيدات Constructors) وكذلك (المهدمات Destructors) ، وأعتذر إن كنت أخرت الحديث حول هاتين النقطتين ولكن كنت أود أن أجملهما في كل من الفئات والتراكيب.

- **المشيدات Constructors :**

يتضح من الاسم أن ما سنتناوله الآن خاص بتشييد أو بدء أو بناء شيء ما. المشيد ما هو إلا إجراء فرعي عادي جدا Subroutine ، ويتم تنفيذ هذا الإجراء بشكل تلقائي عند استخدام الكلمة المحجوزة New أثناء التصريح عن أحد الكائنات كنسخة أو كمؤشر لفئة ما كما يلي :

```
Dim MyObject As New Class1
```

عند استخدامك للكلمة New فهذا يعني أنك تستدعي تنفيذ الدالة Constructor وهي الدالة التي تعمل تلقائيا مع تشغيل أي نسخة من البرنامج ، وعلى ذلك سنذكر المثال الآتي ... إذا قمنا بالتصريح عن الكائن الأتي بالاسم CN كنسخة من الفئة System.Data.SqlClient.SqlConnection كما يلي :

```
Dim CN As New System.Data.SqlClient.SqlConnection("Data Source = MyServer
; Initial Catalog = MyDatabase ; Persist Security Info = True ; User ID =
AhmedNegm ; Password = 0020119777244")
```

وهذا بدلا من كتابة نفس الكود بالشكل التالي :

```
Dim CN As New System.Data.SqlClient.SqlConnection
CN.ConnectionString = Data Source = MyServer ; Initial Catalog =
MyDatabase ; Persist Security Info = True ; User ID = AhmedNegm ;
Password = 0020119777244"
```

مما يعني ذلك أنك تستطيع تمرير نص جملة الاتصال أثناء تصريحك أو إسنادك لنسخة من الفئة المذكورة للكائن CN ، وهذا يتحكم فيها المشيد Constructor ، حيث أن الفئة المذكورة بها الإجراء الفرعي New والذي يمكن عن طريقه تمرير نص جملة الاتصال بدلا من استخدام الخاصية ConnectionString كسطر على حده ... سأروي عليك مثالا بسيطا عن المشيدات ، افترض انك تريد من مستخدم إحدى الفئات التي قمت أنت بصناعتها أنه إذا أخذ نسخة من الفئة تظهر له رسالة كترحيب مثلا أو ما شابه ذلك ، فسيكون ذلك كما يلي :

```
Public Class MyVBClass

    Public Sub New()
        MsgBox("Welcome to .." & vbNewLine & "      SMART SOFT ... For
Spftware Industry !!!", MsgBoxStyle.Information)
    End Sub
```

```
End Class
```

وكذلك يمكنك استخدام المشيد أو الإجراء New في تمرير معاملات إلى الفئة في بداية أخذ نسخة منها ، يمكنك الاطلاع على المثال الآتي الذي يوضح كيفية استخدام أكثر من مشيد وبأكثر من وظيفة ، المثال على الرابط التالي :

< http://www.vb4arab.com/vb/uploaded/13807/01252114057.rar >

- ## المهدمات Destructors :

إذا كنت قد استوعبت ما تم ذكره في المشيدات ، فكن على علم بأننا بصدد عكس ما سبق وأن ذكرناه ، ويتم تنفيذ ما هو داخل المهدم في حال إنهاء استخدامك للكائن Object ، ولنفترض أنك مثلا تريد أن تصدر رسالة وداع أو إصدار صوت Beep عند إنهاء استخدام الكائن أو ..... الخ. ولكن الوضع يختلف قليلا من حيث كتابة الإجراء المهدم فيكون نصه كالتالي :

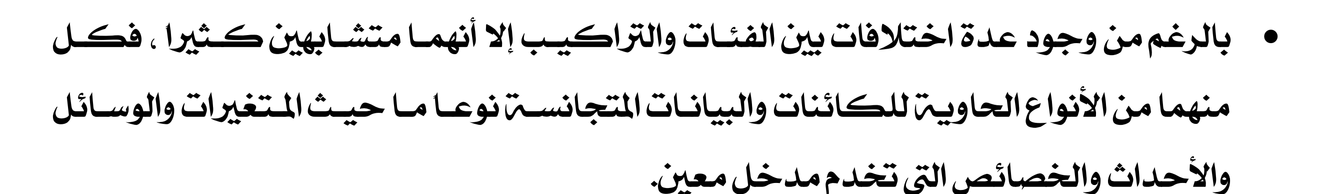
```
Public Class Class1

    Protected Overrides Sub Finalize()
        MsgBox("GoodBye ... !!" & vbNewLine & "Now !! object was
terminated.", MsgBoxStyle.Information)
    End Sub

End Class

                    ** تم اضافة الإجراء المهدم إلى المثال السابق ذكره **
```

لمحات حول Classes و Structures :



- بالرغم من وجود عدة اختلافات بين الفئات والتراكيب إلا أنهما متشابهين كثيرا ، فكل منهما من الأنواع الحاوية للكائنات والبيانات المتجانسة نوعا ما حيث المتغيرات والوسائل والأحداث والخصائص التي تخدم مدخل معين.
- أحدثكم بسر ... غالبا وبنسبة ٩٥٪ من المطورين تقريبا - إن لم يكن أكثر - يستخدمون الفئات Classes وقل من يتعامل مع التراكيب Structure ، ويعتقد بعض المفكرين في مجال صناعة وتطوير البرمجيات أن هذا يرجع للحداثة النسبية التي تتسم بها الـ Structures وكذلك الألفة القوية بين المطورين والفئات Classes ، وكذلك من ضمن الأسباب الأخرى هو عدم اتسام التراكيب Structures بالوراثة وما يصاحبها من مميزات.
- من ضمن الاختلافات بين الـ Classes و Structures هو نوع كل منهما كما تم توضيح ذلك سابقا على أن الفئات من النوع Reference Types أما التراكيب من النوع Value Types ، وهذا بالطبع يعطي جوهرا خاص لكل منهما.
- من ضمن الاختلافات أن الفئات غير إجبارية التشييد ، أي أنه لست مجبرا أن تقوم بعمل مشيد أو أكثر للفئة وكذلك الحال بالنسبة للمهدم ، وكذلك من الممكن إذا قمت بعمل المشيد تستطيع جعل كافة المعاملات من النوع الاختياري Optional أو يمكنك عدم وضع أية معاملات من الأساس في الإجراء New وهو بدوره المشيد للفئة. نأتي للتركيب وهو بعكس ذلك ، ولكنه من سماته أنه يسمح بوجود إجراء مشيد وبنفس الطريقة مثل الفئات تماما ولكن الاختلاف يكمن فيما يلي :
    - يمكنك إضافة المشيدات في الفئات بدون أية معاملات إطلاقا أو أن تكون كل المعاملات من النوع الاختياري Optional كما ذكرنا ، بعكس التراكيب التي يجب عليك عند إضافة مشيد لها أن تضيف معامل واحد على الأقل ومن أي نوع كان.
    - وكذلك لا يمكنك إنشاء مشيد بمعاملات اختيارية أي لا تستطيع كتابة إجراء Constructer لإحدى التراكيب وتكون كافة المعاملات لهذا المشيد من النوع

الاختياري Optional ولكن يجب على الأقل وجود معامل من النوع الإجباري NotOptional Argument.

- أيضا هناك تشابه في استخدام كلمة With كما في الفئات.نعم يمكنك ذلك بكل بساطة أثناء التعامل مع التراكيب Structures ، دقق النظر في الكود التالي حتى تتضح الرؤية:

```
Dim SomeOne As New Person With {.FirstName = "Ahmed", .LastName = "Negm"}
```

- مما سيثير عقلك ، أنك بالرغم من قدرتك على إنشاء تركيب Structure بدون استخدام الكلمة New ، إلا أنك لا تستطيع استخدام كلمة With كما في الكود السابق بدون استخدامك للكلمة New كما ورد في الكود السابق تماما.
- مما سيثير جنونك هنا .. أن الـVisual Basic دائما يسمح بشكل افتراضي استخدام مشيد فارغ المعاملات Empty Constructor ، ولكنك لا تستطيع فعل ذلك يدويا. أقصد أنك إذا لم تقوم بكتابة الإجراء المشيد للـStructure فسيقوم الـVisual Basic بإنشاء إجراء مشيد فارغ المعاملات بشكل تلقائي وغير مرئي ، ولكنك كما ذكرنا لن تستطيع إنشاء مشيد فارغ المعاملات بشكل يدوي.
- من أوجه الاختلاف كذلك هو أنك لن تجد أثناء تعاملك مع الـStructure إمكانية إسناد قيم للمتغيرات كقيم ابتدائية أو افتراضية أثناء التصريح عنها كما هو الحال في الفئات ... انظر في الكود التالي ولاحظ ما كتب باللون الأخضر :

```
'Class
Public Class Person
    Public FirstName As String = "<Unknown Name>"    'Initialization value allowed
    Public LastName As String = "<Unknown Name>"     'Initialization value allowed

    'Empty constructor allowed
    Public Sub New()
        'No Code
    End Sub

    'Two-parameters constructor allowed
    Public Sub New(ByVal First_Name As String, ByVal Last_Name As String)
        FirstName = First_Name
        LastName = Last_Name
    End Sub

    'Optional parameters constructor allowed
```

```
    Public Sub New(Optional ByVal FName As String = "Ahmed", Optional
ByVal LName As String = "Negm")
        FirstName = FName
        LastName = LName
    End Sub

End Class

'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
'@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'

'Structure
Structure Person
    Public FirstName As String = "<Unknown Name>"   'Initialization value NOT allowed
    Public LastName As String = "<Unknown Name>"    'Initialization value NOT allowed

    'Empty constructor NOT allowed
    Public Sub New()
        'No Code
    End Sub

    'Two-parameters constructor allowed
    Public Sub New(ByVal First_Name As String, ByVal Last_Name As String)
        FirstName = First_Name
        LastName = Last_Name
    End Sub

    'Optional parameters constructor NOT allowed
    Public Sub New(Optional ByVal FName As String = "Ahmed", Optional
ByVal LName As String = "Negm")
        FirstName = FName
        LastName = LName
    End Sub

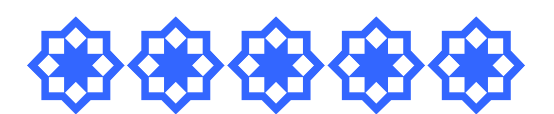
End Structure
```

قم بتجربة الكود السابق داخل محرر اللغة Visual Basic وانظر بنفسك لمواضع الخطأ. وإلى ماذا سيشير إليك المحرر وماذا سينص عليك.

- من البديهي أنك إذا فهمت الفرق بين الـ Classes و Structures ، فعندئذ يمكن فقط تحديد متى وأين يمكنك استخدام أي منهما في تطبيقاتك.

ثانياً: البرمجة كائنية التوجه

Object-Oriented Programming



عندما نتحدث عن الـ OOP أو ما يسمى بالـ (**Object-Oriented Programming**) ، فهنا نحن بصدد الحديث عن البرمجة الكائنية التوجهية او ما يسمى بالبرمجة الموجهة للكائنات. يتناول هذا الموضوع باذن الله تعالى مميزات وسمات ومبادئ الـ OOP والكامنة في الثلاث مبادئ الأساسية كما يلي :

- التغليف ... Encapsulation
- الوراثة ... Inheritance
- التعدد أو تعدد الأوجه ... Polymorphism

هذا بالإضافة لعدة مميزات أو مبادئ أخرى منبثقة من تلك الأساسية التي سبق وأن ذكرناها أعلاه. تعال معي أخي القارئ لنتناول تلك المبادئ الأساسية وتلك الفرعية المنبثقة منها كما هو أدناه.

التغليف (Encapsulation)

الواجهة العامة للفئات Class's Interface تتكون مما سبق ذكره وهي الخصائص وكذلك الطرق والأحداث ، وعندما أتحدث عن الواجهة العامة أقصد بذلك الوظائف والأحداث والخصائص التي تظهر للمستخدم خارج الفئة.

```
Dim CN As New SqlClient.SqlConnection
CN.Open()
```

لو نظرنا في هذين السطرين السابقين فسيتضح ما يلي ... أولا الفئة المسماة SqlConnection عبارة عن فئة مشتقة (فرعية) من مكتبة أو فئة كبرى مسماة SqlClient ، أما عن الكائن CN فهو كائن يحمل خصائص الفئة SqlConnection فإذا كتبت في محرر VB .NET اسم الكائن CN ثم ضغطت علامة (.) ستظهر كافة الخصائص والطرق الخاصة بالفئة SqlConnection ولكن ستظهر أسماء تلك الخصائص والطرق وليس الأكواد التي كُتبت بها تلك الطرق والخصائص.

تعال نضرب مثال آخر .. في مثالنا السابق عن الفئة المسماة Job فهناك طريقة أو أجراء بالاسم AssignJob ولكن من الطبيعي لتنفيذ هذه الوظيفة السابق ذكرها أن يكون هناك دالة أثناء التنفيذ بأن يقوم البرنامج بفحص مؤهلات أحد الموظفين للقيام بمتابعة تعيين العميل المتقدم للوظيفة كي ينفذ الوظيفة الرئيسية وهي التعيين ولتكن مثلا اسم الدالة الفرعية او الثانوية هذه بالاسم FindQualifiedEmployee. وفيما سبق قصدت بأن الدالة المسماة FindQualifiedEmployee غير ظاهرة لمستخدم الـ Class لذا فهي تم تعريفها أو التصريح عنها على أنها خاصة Private أما عن الإجراء AssignJob فتم التصريح عنه بالمفتاح Public. طبقا لما سبق فإن مستخدم الفئة Job ستظهر له الطريقة AssignJob عند التصريح عن كائن كنسخة من الفئة ، أما الدالة FindQualifiedEmployee فهي خاصة بالفئة فقط ولا تظهر للمستخدم الفئة.

بكل اختصار في حديثنا عن التغليف أو الـ Encapsulation ، أنت تتحكم فيما يظهر لمستخدم الفئة وفيما لا يظهر من خصائص وطرق وأحداث. وما يظهر فهو يؤدي الوظيفة التي وُجدت من أجلها الفئة (Class) ، أما ما لا يظهر للمستخدم من أحداث وخصائص وطرق فهو يخدم ما يظهر مما سبق.

يمكنك الاستفادة من مميزات التغليف Encapsulation بعيدا عن الفئات. في العِقد السابق قبل اعتماد بعض اللغات لمبادئ الـ Object-Oriented كان المبرمجين يقومون بصناعة مكتبات الكود أو تطبيقات معينة وتدعم أيضا التغليف وبنفس المفهوم الحالي ولكن كان آنذاك بمعنى آخر. على سبيل المثال : لو لدينا مكتبة تحتوي على وظائف ودوال لعلم المثلثات Trigonometry مثل الدوال Sines أو Cosines أو Tangents أو

Arctagents ... الخ ، ولتنفيذ هذه الحسابات يجب أن تحتوي المكتبة على وظائف ودوال مساعدة خفية تساهم في الوصول لنتائج الدوال الأساسية السابق ذكرها.

الوراثة (Inheritance)

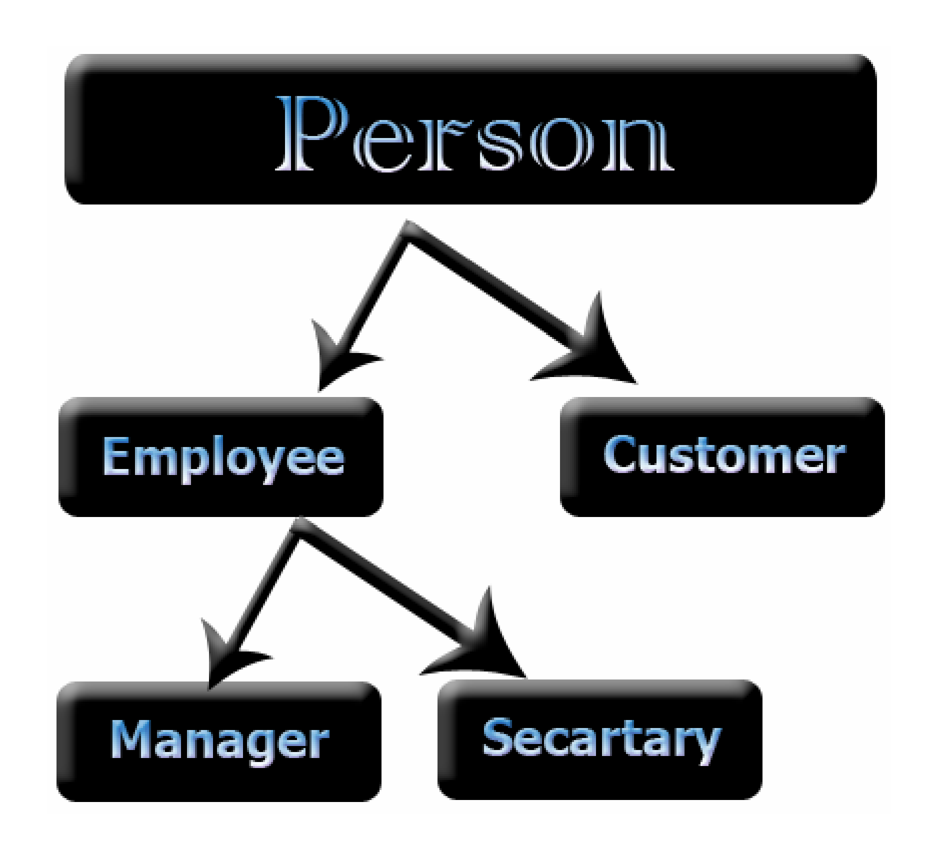
الوراثة ضمن مبادئ الـ OOP عبارة عن اشتقاق فئة من فئة أخرى. وتكون الفئة الجديدة المشتقة (Child Class or Derived Class or SubClassing) تحتوي على كافة خصائص وأحداث وطرق الفئة الأب Parent Class. هذا بالإضافة إلى انك تستطيع إضافة أحداث وخصائص ودوال وإجراءات جديدة إضافة لما حصلت عليه عن طريق الوراثة من الفئة الأب من الخصائص والأحداث والدوال والإجراءات.

على سبيل المثال : لدينا فئة بالاسم Person ولهذه الفئة بعض المتغيرات كـ (FirstName , LastName , Age , Address , EMail) ، ومثلا هناك وظيفة اسمها DialPhone وهذه تختص بقيام التطبيق بالاتصال تليفونيا بالشخص وكذلك وظيفة أخرى اسمها SendMail وهذه تمكن المستخدم من إرسال بريد إلكتروني للشخص.

إذا أردت إنشاء فئة جديدة بالاسم Employee فهنا ستجد أن كل موظف ما هو إلا شخص عادي لديه الخصائص والوظائف السابق ذكرها (FirstName , LastName , Age , Address , EMail) وكذلك DialPhone و SendMail. وهنا إذا لم تكن تعرف للوراثة معنى فستقوم بإنشاء تلك الخصائص أو المتغيرات والوظائف ثانية من جديد ، وستضيف عليها طبعا بعض الخصائص التي تخص الموظف مثل (Salary , WorkHours , WorkAddress , WorkPhone , EmployeeID , OfficeNumber) ، وتريد كذلك استخدام الوظيفة DialPhone ولكن بشكل آخر وبأسلوب أخر.

وكذلك ستظل المشكلة قائمة إذا أردت عمل فئة جديدة بالاسم Manager ، فستحتاج كافة الخصائص والوظائف الموجودة في كلتا الفئتين Person وكذلك Employee

وستضيف عليها لاحقا بعض الخصائص والأحداث والوظائف وذلك لان كل مدير ما هـو إلا موظف وما كل موظف إلا شخص عادي.



الشـكل السـابق كـان مثـالا لمشـكلتنا السـابق تفصـيلها. مـن مميـزات الوراثـة والتوريـث الأساسية هي إعادة استخدام الكود مـرة أخـرى دون إنشـاؤه أو حتـى إعـادة كتابتـه مـرة أخـرى. ومعنى هذا أنـك مـثلا عنـدما سـتكون بصـدد إنشـاء الفئـة المسـماة Employee سـتقوم بإنشـاء (FirstName , LastName , Age , Address , EMail) وكـذلك الوظـائف والطـرق السابق ذكـرها. ولكـن بمجرد أخذك قرار الوراثة من الفئة الأب Person سـتحل هـذه المشـكلة بالكـامل.

❁ التوريث المتدرج Inheritance Hierarchies

إعادة استخدام الكود في التوريث لا يحميك من الخطأ في إعادة كتابة الكود فحسب ولكن كذلك يجعل تطوير الفئة أو التطبيق بصفة عامة اسهل من ذي قبل. بفرض مثلا أنك تقوم بتطوير مجموعة من الفئات كما في الشكل السابق ، وبعد انتهائك من الفئات Person كفئة أساسية Parent Class ، وكذلك الفئات المشتقة مثل Employee و Manager و Customer و Secretary كفئات مشتقة Subclasses ... أردت أن تضيف الخاصية BirthDate ، فلو ابتعدت عن مبادئ التوريث التي جاءت بها البرمجة الموجهة للكائنات فستقوم بإضافة هذه الخاصية إلى كل الفئات السابق ذكرها ، أما مع OOP ومع Inheritance فما عليك إلا أن تضيف هذه الخاصية في الفئة Person لأنها الفئة الأب ومن ثم للفئات الأخرى يورثوا خواص الفئة الأب وانتهى الموضوع وتم حل المشكلة.

كمثال آخر ، بفرض أنك تريد تعديل أو حذف أحد الخصائص أو الوسائل ولتكن مثلا داخل خاصية FirstName ، فما عليك إلا أن تقوم بتعديل الخاصية المذكورة في الفئة الأب وهي Person ولا عليك أن تقوم بالتعديل في كل الفئات التي تحمل نفس الخاصية.

❁❁ معلومة ❁❁

تدعم بعض اللغات التوريث المتعدد ، بمعنى أنه يمكنك في فئة مشتقة من أكثر من فئة أب. فلنفترض أن هناك فئة بالاسم Car وتحتوي على بعض الخصائص والوسائل والأحداث الخاصة بهذه الفئة ، وهناك فئة أخرى بالاسم House ولهذه الفئة أيضا ما يميزها من الخصائص والأحداث. ولو استخدمنا التوريث المتعدد Multiple Inheritance فسنحصل على فئة أخرى ولتكن بالاسم MotorHome ويحمل مميزات كلتا الفئتين Car و House.

أنا شخصيا – بعقلي الضيق الفكر- غير متخيل الفئة الناتجة لأن الفئات الأساسية غير متجانسة ، ولكن Microsoft أشفقت عليَ وجعلت NET. Visual Basic لا يدعم التوريث المتعدد وجعلت الفئة لا تورث من أخرى إلا مرة واحدة فقط. إذا احتجت لاستخدام التوريث المتعدد فعليك أن تستخدم الواجهات Interfaces ، فبدلا من تعريف أكثر من Parent Class فقم بتعريف أو التصريح عن أكثر من Parent Interface وعندها يمكنك إنشاء فئة مشتقة

تنفذ أو تطبق داخلها أكثر من واجهة Interface كما يحلو لك. وللعلم لا تورث الفئة أية أكواد من الواجهات ولكن من اسمها ستأخذ فقط واجهة وتضيفها على الفئة.

❀❀ معلومة أخرى ❀❀

يمكنك استخدام الكلمات الدليلية MustInherit وكذلك NotInheritable ، فالأولى تمنع استنساخ الفئة باستخدام عبارة كهذه :

```
Dim MyObject As New Class1 'Not Okay .. because of this class must inherit.
```

ودائما تجبر مستخدم الفئة بوراثتها في فئة أخرى ، وبذلك يمكنك استخدام الكود السابق مع الفئة المشتقة ولتكن مثلا بالاسم Class2 وليس الفئة الأب Class1 وبذلك يمكن أن يكون الكود كما يلي :

```
Dim MyObject As New Class2 'Okay .. this is the derived class.
```

وبهذا أنت قد تحكمت في الفئة كما تحب ولكن طبعا أنت تتساءل كيف سأصل للوسائل ومكونات الفئة من متغيرات وما شابه ، وهنا سأجيبك قائلا بأنك تستطيع ذلك كله عن طريق الفئة الجديدة أو المشتقة Derived Class.

أما عن الكلمة الدليلية الثانية NotInheritable فهي تمنع اشتقاق الفئة داخل فئة أخرى باستخدام العبارة :

```
Inherits Class1
```

ولكن يمكنك وقتها استنساخ الفئة والتعامل معها بشكل طبيعي جدا ، ولكن بعيدا كل البعد عن مفاهيم الوراثة أو الاشتقاق.

ويمكنك تحميل المثال الخاص بهذه الجزئية من الرابط التالي ، وقم بنفسك بتفحص الأخطاء التي ستظهر نتيجة الكود المكتوب :

< http://www.vb4arab.com/vb/uploaded/13807/01251733802.rar >

ولكن يجب أن تعلم أن MustInherit تعمل كعكس للكلمة الدليلية NotInheritable وهذا يتضح من وظائفهما كما سبق وأن ذكرنا.

✿ التوريث وإعادة القيادة Overriding

إذا قمنا باشتقاق الفئة Employee من الفئة Person ، فسنقوم بإضافة خواص ووسائل جديدة للفئة Employee كما تم الذكر في أول الموضوع ، ولكن من ضمن المشاكل التي واجهتنا هي أن الوظيفة DialPhone تقوم بالاتصال بالشخص عن طريق المودم ولكن تتصل بتليفون المنزل ، أما في الفئة Employee نحتاج نفس الوظيفة وبنفس إمكانياتها باستثناء رقم التليفون ، فنحن نريد استبداله بتليفون العمل. لو بعدنا قليلا عن OOP وكذلك الـ Overriding فسنقول أن الحل هي إنشاء الوظيفة من جديد ، ولكن مع OOP فسنستخدم إحدى مميزاتها الفرعية وهي إعادة القيادة للوظائف والوسائل ، ولكن من شروط استخدام إعادة القيادة هي أن يكون الإجراء الأساسي في الفئة الأب معرف أو مصرح على أنه قابل لإعادة القيادة من مكان آخر. فيجب أن يكون الإجراء في الفئة الأب معرف كما يلي :

```
'Dial the phone using (Phone) property.
Public Overridable Sub DialPhone()
MsgBox("Dial :  " & Me.Phone)
'Other Code
End Sub
```

وعند الاشتقاق أو التوريث وإقرار تولي القيادة من الفئة المشتقة فسيكون الإجراء معرف أو مصرح عنه كما يلي :

```
'Dial the phone using (WorkPhone) property.
Public Overrides Sub DialPhone()
MsgBox("Dial :  " & Me.WorkPhone)
'Other Code
End Sub
```

تتيح لك إعادة القيادة استخدام إجراء بنفس الاسم مرة أخرى ولكن تحت أكواد جديدة ووظائف من الممكن أن تكون مغايرة تماما للوظيفة الموجودة في الفئة الأب ولكن بشرط أن تكون الوظيفة معرفة في الفئة الأب على النحو التالي `Public Overridable Sub` ويقصد بها أي ان هذا الاجراء قابل لإعادة تولي القيادة من مكان آخر ، ولتولي القيادة من مكان آخر فيجب التصريح عن الوظيفة في الفئة المشتقة كما يلي `Public Overrides Sub` ، وبهذا تكون قد أخفيت هذه الوظيفة في الفئة الأب وأظهرتها من جديد في الفئة المشتقة وبشكل جديد وذلك كما رأينا في مثالنا السابق.

❁❁ ملحوظة ❁❁

كي تعطي أمر بالوراثة من فئة أب فما عليك إلا كتابة ما يلي :

```
Inherits Person
```

حيث Person هي الفئة الأب المشتق منها خصائصها داخل الفئة المشتقة ولتكن كما ذكرنا الفئة Employee ... يمكنك النظر في المشروع التالي لرؤية الأكواد المتعلقة بالوراثة وكذلك إعادة القيادة ، ولكن عند تحميلك للمثال ستجد به فئة بالاسم Person وفئة أخرى بالاسم Employee ونموذج بالاسم Form1 ، قم بتعريف كائنات جديدة كنسخة من الفئات السابق ذكرها وشاهد مميزات إعادة القيادة وكذلك مميزات الوراثة بنفسك.

< http://www.vb4arab.com/vb/uploaded/13807/01248952080.rar >

بما أننا قد ذكرنا إعادة القيادة Overriding فتعال نذكر شكل آخر من إعادة القيادة وهو Shadow.

❁ إعادة القيادة والظل Shadows

تحت العنوان الجانبي السابق (التوريث وإعادة القيادة Overriding) قمنا بطرح مشكلة وهي إعادة كتابة إجراء بنفس الاسم لكن مع اختلاف الوظيفة التي خلِق من أجلها. هنا تظهر مشكلة جديدة : ماذا لو أننا لدينا الفئة Person بما فيها للإجراء DialPhone ولكنه غير قابل لإعادة القيادة من مكان آخر ، وكذلك الفئة Person تم تغليفها وجلبها للمشروع في صورة DLL File وعندئذ لا يمكن التعديل فيما ورد بها وكذلك الإجراء المسمى DialPhone في الفئة الأب غير قابل لإعادة تولي القيادة يعني أنه لم يصرح عنه كـ Overridable Sub وبالتالي لا يمكننا إعادة تولي قيادته طبقا لم تم شرحه في العنوان الفرعي السابق (التوريث وإعادة القيادة) ، ولكننا في نفس الوقت نريد إعادة تولي القيادة للإجراء DialPhone وتغيير بعض خصائصه وعدم إظهار نفس الإجراء في الفئة الأب ... ماذا سنفعل ؟؟؟. باستخدام للكلمة المفتاحية Shadows أثناء تصريحك عن الإجراء في الفئة المشتقة بهذا تكون قد فعلت أو قمت بما تقوم به Overridable Sub دون أن يكون الإجراء قابل لإعادة تولي القيادة من مكان آخر ،

فكلمة Shadows تخفي الإجراء في الفئة الأب وتظهر فقط الإجراء بنفس الاسم في الفئة المشتقة.

❁❁ ملحوظة مهمة ❁❁

بفرض أن هناك في الفئة الأب متغير نصي بالاسم (FirstName) ، وفي الفئة المشتقة يوجد إجراء فرعي بالاسم (FirstName) .. هنا لا تستخدم shadows لان هذا سيسبب تعارض لان أنواع الكائنات غير متجانسة فهذا متغير وهذه وسيلة Method.

❁ وراثة الواجهات Interface Inheritance

قد يعترض معي البعض – وقد اعترض احدهم بالفعل – في صياغة هذا العنوان وكيف قد هممت بالحديث عن الواجهات Interfaces تحت بند الوراثة أو التوريث Inheritance ، ولكن دعونا مما اعتدنا عليه ونترك القضية كاملة للمنطق فنقول ما يلي : " أنا شخصيا مع من يقول بأني مخطئ من جهة واحدة وهي أنه عند استدعاء الواجهة أو تنفيذها داخل الفئة فإننا لا نستخدم الكلمة Inherits ولكنك تستخدم كلمة أخرى كما سنعلم لاحقا ، وكذلك يمنع الوراثة من أكثر من كائن ولكن في الواجهات يمكن تطبيقها أكثر من مرة داخل الفئة مما يؤيد وجهة النظر هذه. وعلى الجهة الأخرى ، فرأيي الشخصي في هذا الصدد هو أنك فعلا تقوم بوراثة التصريحات التي تقوم بالتصريح عنها داخل الواجهة إلى داخل الفئة ، ولكن هنا مسموح بالوراثة المتعدد فقط في حالة الواجهات. وبحمد الله تعالى قد جاء الكتاب الذي اتخذته مرجعا لي أثناء إعداد هذا الكتاب والذي ذكرت عنوانه في مقدمة الكتاب مؤيدا لما اعتقدت فيه". ولكن دعونا من هذا ونذهب إلى حيث نستفيد.

تحدثنا سالفا عن اشتقاق او وراثة فئة من أخرى وفيها تورث الفئة المشتقة كافة خصائص ووسائل وأحداث الفئة الأب وكذلك أيضا وراثة الكود المكتوب بالداخل كاملا. يمكنك Visual Basic .NET من تعريف أو التصريح عن الواجهات ، والواجهة ما هي إلا واجهة لفئة ما واسمها خير دليل على هذا وهي لا تحتوي على أية أكواد للتنفيذ. بعد تصريحك عن واجهة

جديدة يمكنك استخدام الكلمة <InterfaceName> Implements وذلك كي تشتق أو تستدعي كافة الوسائل والخصائص والأحداث التي تم التصريح عنها داخل الواجهة.

كي تتضح لديك الرؤية أكثر من ذلك ، اتبع ما يلي ... قم بالتصريح عن هذين المتغيرين كما هو موضح أدناه :

```
Dim CN1 As New System.Data.OleDb.OleDbConnection
'-------------------------------------------------
Dim CN2 As New System.Data.SqlClient.SqlConnection
```

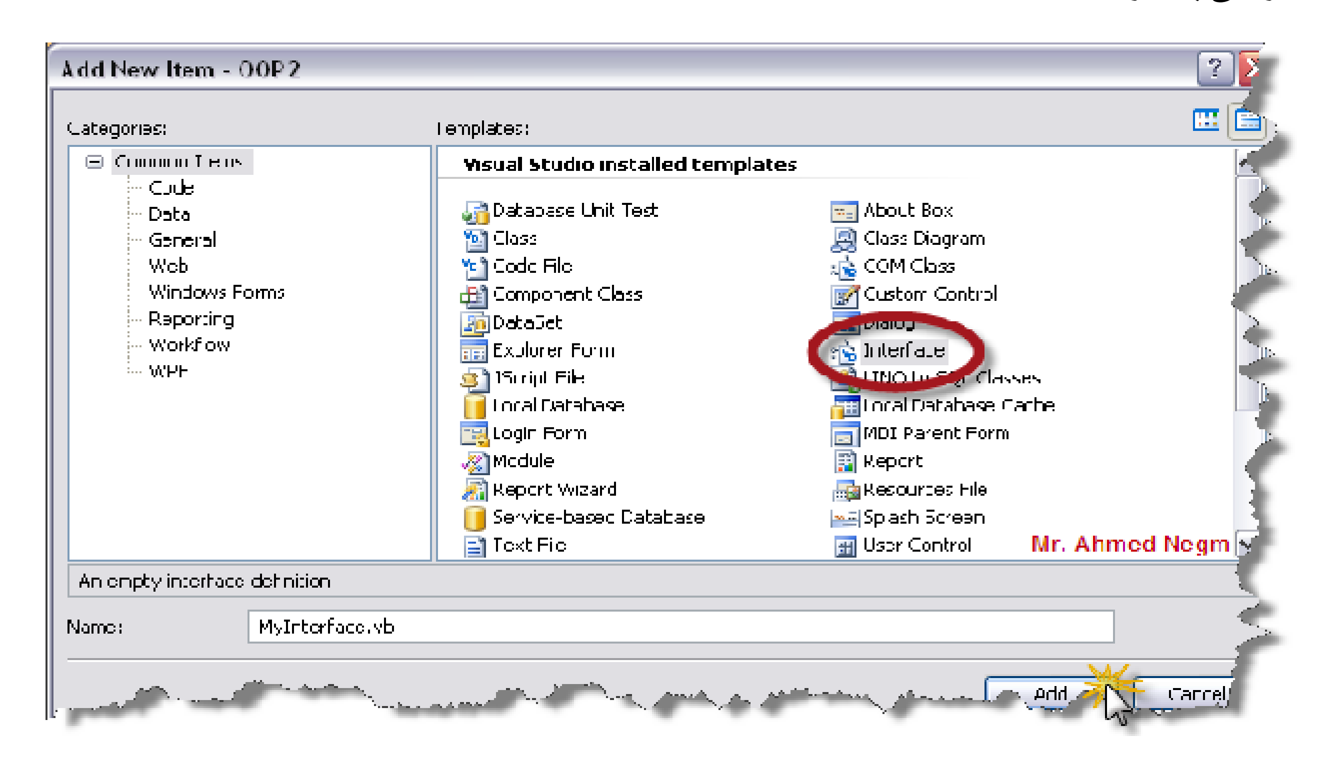
العجيب في هذين التصريحين أنك إذا نظرت في الوسائل والخصائص المصاحبة للكائن CN1 ستجدها بنفس الترتيب ونفس الهجاء بالنسبة للكائن CN2 ، ولكن مع اختلاف الوظيفة ... يعني مثلا الوظيفة Open فهي مصاحبة لكلا الكائنين ولكنها مع CN1 تتصل بأي قاعدة بيانات حسب جملة الاتصال المررة إليها Connection String ، أما مع الكائن CN2 فهي لا تتعامل إلا مع محركات قواعد بيانات MS SQL Server فقط بالرغم من أن اسم الوظيفة واحد في كلا الكائنين ... هنا يتضح معنى الواجهات.

سأوضح لك مرة أخرى وبمثال آخر .. بفرض أنك مدير مشروع برمجي معين وليكن هذا المشروع خاص بصناعة مكتبة DLL للتعامل مع ملفات MS OFFICE ، وقد قررت كمدير مشروع أن تقوم بعمل أربعة فئات 4 Classes ، الأولى منهم لـ MS Word والثانية لـ MS Access والثالثة لـ MS PowerPoint والرابعة لـ MS Excel ، وتم الاتفاق على عدة وظائف مثلا كالآتي :

- Method >> OpenFile
- Method >> DeleteFile
- Property >> FileName
- Method >> FileSize

وهذا فقط كان على سبيل المثال .. تعال معي نرجع سويا إلى موضوعنا الأساسي ، أنت كمدير مشروع كما ذكرنا سالفا أنك تريد تواجد هذه الخصائص والوسائل في كل فئة من الأربعة فئات المرجو إنشائها. هل الحل أنك تقوم بالنسخ واللصق ومن ثم تغيير الأكواد ، أم الإملاء على المطورين ومن ثم تجد الأخطاء ؟؟؟ ... هنا سنلجأ إلى استخدام الواجهات فهي من ستحل الموضوع.

من القائمة Project اختر العنصر Add Component ثم اختر العنصر Interface كما هو موضح بالصورة أدناه :



قد قمت بالتصريح عن الوسائل والخصائص السالف ذكرها في الواجهة كما يلي ... ولا زلت أكرر أننا لا نكتب أية أكواد داخل الواجهة ، فقط نقوم بالتعريف والتصريح عن مكونات الفئة كما يلي :

```
Public Interface MyInterface

    'This subroutine to open the file by using [FileName] property value
    Sub OpenFile()

    'Use this property to specify file path or read the file path to open it
or delete it
    Property FileName() As String

    'This subroutine to delete the file by using [FileName] property value
    Sub DeleteFile()

    'This function used to return the file size by using [FileName] property
value
    Function FileSize() As Double

End Interface
```

أنا فقط قمت بعمل مثال عابر ومبسط ، فيمكنك عزيزي القارئ أن تقوم بالتصريح عما تحب وكيف تحب ومتى تحب ... قم بالتصريح مثلا عن متغيرات عادية وأحداث ووسائل وخصائص ، أما

أنا لضيق الوقت أردت إيصال الفكرة فقط. أنا متشوق بالفعل حتى أريك كيف يتم وراثة هذه الواجهة داخل الفئات التي قمنا بإنشائها سابقا ، فكل ما عليك إلا الدخول داخل كل فئة وتكتب هذه العبارة :

```
Implements MyInterface
```

ستجد نفسك في الفئة مثلا الخاصة بالـ MS Excel قد حصلت على النتيجة التالية كوراثة حقيقية لما تم التصريح عنه في الواجهة MyInterface:

```
Public Class Excel
    Implements MyInterface

    Public Sub DeleteFile() Implements MyInterface.DeleteFile

    End Sub

    Public Property FileName() As String Implements MyInterface.FileName
        Get

        End Get
        Set(ByVal value As String)

        End Set
    End Property

    Public Function FileSize() As Double Implements MyInterface.FileSize

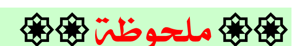
    End Function

    Public Sub OpenFile() Implements MyInterface.OpenFile

    End Sub
End Class
```

هذه هي بالفعل شكل الفئة بعد وراثة الواجهة وما عليك إلا توزيع الهيكل العام لهذه المكتبة DLL على المطورين في المشروع وهم يقوموا بكتابة الكود فقط داخل الوسائل والخصائص والأحداث المصرح عنها ... يمكنك تحميل المشروع من الرابط التالي :

< http://www.vb4arab.com/vb/uploaded/13807/01250864124.rar >



كما ذكرنا سابقا أن Visual Basic .NET لا يسمح لك بالوراثة من أكثر من فئة ، ولكن داخل الفئة الواحدة يمكنك وراثة أكثر من واجهة Interface.

❁ الكلمات المفتاحية Access Modifiers

- **Public** : يجعل المتغير أو الوسيلة معرفة على مستوى الفئة ككل وكذلك على مستوى المشروع بالكامل ، كما يمكنك الحصول على هذا المتغير أو هذه الوسيلة إذا قمت بأخذ نسخة Instance من الفئة في أي مكان آخر على اختلاف نوع المشاريع سواء WindowsApplications أو ClassLibrary ... الخ. كما أن هذا المتغير أو الوسيلة من النوع Public تظهر في حالات الوارثة في الفئات المشتقة.
- **Friend** : يجعل الإجراء معرف على مستوى الفئة ككل وداخل المشروع الحالي بصفة عامة ، لكن اذا تم دمج الفئة في ملف DLL وأخذ نسخة لكائن من الفئة فإن الاجراءات من هذا النوع لا تظهر.
- **Private** : يجعل الإجراء معرف على مستوى على الفئة فقط ، ولا يظهر في أي مكان خارجها.
- **Protected** : يشبه الـ Private تماما ولكن يختلف عنه في نقطة ، ألا وهي أنه في حالة التوريث فإن الفئة المشتقة Derived Class يظهر بداخله الإجراءات والمتغيرات من النوع Protected ، ولكن إذا أخذت نسخة من الفئة الأب Base Class أو ما يسمى بالـ Parent Class فلن تظهر المتغيرات ولا الإجراءات من النوع Protected.

يمكنك مراجعة المثال التالي ، فهو عبارة عن فئة بالاسم (MyClass) وبها بعض الإجراءات باختلاف الكلمات المفتاحية لكل منها:

< http://www.vb4arab.com/vb/uploaded/13807/31250864124.rar >

تعدد الواجهات (Polymorphism)

من أجمل ما قرأت عن التعدد أو تعدد الواجهات أو الـ Polymorphism كان ذلك في كتاب (Visual Basic 2008) لكاتبه Rod Stephens كان الآتي بالنص :

Roughly speaking, *polymorphism* means treating one object as another. In OOP terms, it means that you can treat an object of one class as if it were from a parent class.

فعلا هكذا ، أثناء تعرضك للتعدد أو الـ Polymorphism فأنت يمكنك معاملة الكائن كما لو كان كائن آخر .. على سبيل المثال فلنفترض افتراضاتنا السابقة وهي أنه لدينا الفئة المسماة Employee وكذلك الفئة Customer وهما مشتقان من الفئة الأب المسماة Person. من هنا يمكنك معاملة الكائنات المصاحبة للفئتين Employee وكذلك Customer كما لو انك تتعامل مع كائنات الفئة الأب Person وذلك لأنهم بكل بساطة للفئة الأب وليس الابن كما يظهر لك. يعني هذا انك ظاهريا تتعامل مع الفئات Customer و Employee ولكن تستخدم هاتين الفئتين كجسر من خلاله تتعامل في الباطن مع كائنات الفئة الأب Person.

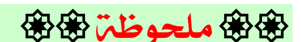
يمكنك الـ Visual Basic .NET من تعيين أو تخصيص قيمة من فئة مشتقة إلى متغير في الفئة الأب. في المثال التالي يمكنك أن تضع كائني Employee أو Customer في متغير من الفئة الأب ... اتبع الكود ثم تابع رباط المشروع في الأسفل :

```
Dim emp As New Employee    'Create an Employee.
Dim cst As New Customer    'Create a Customer.
Dim per As Person          'Declare a Person variable.

per = emp                  'Ok .. An Employee is a Person.
per = cst                  'Ok .. A Customer is a Person.
emp = per                  'Not Ok .. A Person is not necessarily an Employee.
```

يمكنك تحميل المشروع من الرابط التالي :

< http://www.vb4arab.com/vb/uploaded/13807/11250864124.rar >



يمكنك الوصول للمعالم أو المكونات المعرفة الخاصة بنوع المتغير الذي تستخدمه للإشارة إلى كائن آخر. هذا يعني أنك مثلا لو لديك متغير من النوع Person يشير إلى كائن من النوع Employee فإنك فقط تستطيع الاستفادة من معالم الفئة Person وليس لك الحق في الاستفادة مما تم إضافته على الفئة Employee بعد الوراثة.

تعدد أو إعادة التعريف Overloading

ماذا لو أنك قمت بتصميم فئة Class وأردت عمل إجراء فرعي Subroutine بالاسم InsertPersonData ، وهذا الإجراء الفرعي يحتوي على البارامترات أو المعاملات الآتية :

- FirstName
- LastName
- Age
- Address
- EMail
- MobileNumber

المعاملان (FirstName , LastName) يعدان من المعاملات الإجبارية Not Optional Parameter ، أما عن بقية المعاملات فهي اختيارية حسب المتوفر لدى مستخدم الإجراء InsertPersonData ، ولو تخلينا عن مبدأ الـ Overloading ضمن مبادئ الـ OOP فسيكون شكل الإجراء الفرعي كالآتي :

```
Public Sub InsertPersonData(ByVal FirstName As String, ByVal LastName As
String, Optional ByVal Age As Integer = 20, Optional ByVal Address As String
= "", Optional ByVal EMail As String = "", Optional ByVal MobileNumber As
String = "")

'---------------------------
'Your Convenient Code Lines
'---------------------------

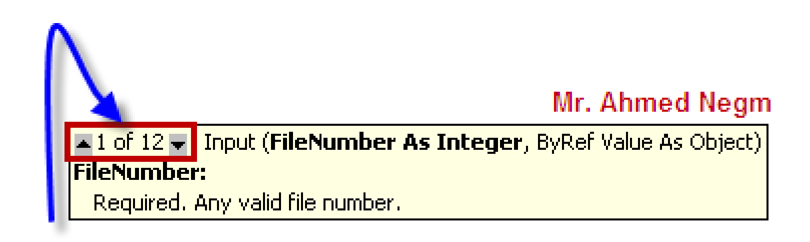
End Sub
```

فقد قمنا في الإجراء السابق بجعل بقية الاختيارات من النوع Optional ولكن للأسف وبكل أسف ستجد أن المعاملات من النوع Optional يجب أن تحتوي على قيم مبدئية وهذا نصا للاعتراض الذي يصدره الـ Visual Basic .NET بالنص الآتي :

Optional parameters must specify a default value

ولو طبقنا هذا الاعتراض في مثالنا الحالي فسنجد مثلا أننا قمنا بتعريف المتغير Age على أنه متغير رقمي من النوع Integer ونوعه اختياري Optional وقيمته الافتراضية = ٢٠ ، وهذا يعني أن المستخدم لهذا الإجراء لن يستطيع الاستفادة من هذا الإجراء لأنه لو لم يقوم بإدخال عمر الشخص فان البرنامج يقوم بشكل افتراضي بوضع القيمة ٢٠ في حقل العمر في حين أن المستخدم للإجراء لا يعرف أو لم يتوفر لديه أية بيانات عن عمر الشخص.

أعتذر لو استطردت كثيرا في وصف المشكلة ، ولكن هنا يأتي دور الـ Overloading ، فهل تعلم أن الـ Visual Basic .NET يعطيك إمكانية تكرار أو إنشاء أكثر من وسيلة Method بنفس الاسم ولكن بشرط اختلاف المعاملات. أي أنك من الممكن مثلا أن تقوم بإنشاء الإجراء السابق المسمى InsertPersonData أكثر من مرة وبنفس الاسم ولكن يختلف في المعاملات المررة إليه. بالطبع أثناء استخدامك للغات الـ NET. لاحظت الآتي :



إذا نظرت إلى ما يشير إليه السهم في الصورة السابقة ستجد الرمز الآتي (1 of 12) وهذا يعني بكل بساطة أن الإجراء أو الوسيلة المسماة Input موجودة داخل الـ NET Framework. اثنتي عشرة مرة ولكن مع اختلاف المدخلات أو المعاملات وهذا تطبيق حي للـ Overloading.

تعال معي نضرب مثلا آخر .. الفئة المسماة Person يظهر فيها إجراءان كإجراءات مشيدة للفئة Constructor Subroutines بالاسم New. الإجراء المشيد الأول لا يأخذ أية معاملات بينما الإجراء المشيد الثاني يأخذ المعاملان المتغيران (FirstName , LastName) كقيم افتراضية

أثناء أخذ نسخة Instance من الفئة. وسيكون شكل الفئة Person بعد كتابة الكود كما يلي :

```
Public Class Person

    Public FirstName As String
    Public LastName As String

    Public Sub New()
        FirstName = "<fname>"
        FirstName = "<lname>"
    End Sub

    Public Sub New(ByVal first_Name As String, ByVal last_Name As String)
        FirstName = first_Name
        LastName = last_Name
    End Sub

End Class
```

وإذا ذهبت للنموذج وقمت بأخذ نسخة من هذه الفئة سيظهر لديك الشكل التالي :



وهنا تتضح فائدة الـ Overloading فبدلا من استخدام المعاملات الاختيارية Optional Parameters وكذلك حالات الشرط IF Conditions ، فنلجأ حينئذ لمثل هذه الطرق والتي تحل مشاكل كثيرة. وفيما يلي مثال صغير عن أخذ نسختين وإسنادهما للكائنات Person_1 وكذلك Person_2 كما يلي :

```
Dim Person_1 As New Person
Dim Person_2 As New Person("Ahmed", "Negm")
```

يمكنك تحميل المثال من الرابط التالي :

< http://www.vb4arab.com/vb/uploaded/13807/21250864124.rar >

ولكن هل جاء بذهنك أخي القارئ ما هي ميكانيكية تحديد الإجراء ، يعني مثلا أنه لدينا ثلاثة إجراءات بالاسم New وبالطبع كما ذكرنا أن لكل منهما معاملاته الخاصة التي تختلف عن المعاملات الخاصة بنظائره ، فكيف أقوم بتحديد التعامل مع الأول أو الثاني أو الثالث ... الخ . هنا يقوم الـ Visual Basic .NET بتحديد الإجراء حسب نوعية وعدد الإجراءات المررة إليه وليس عليك أية التزامات تجاه هذه الجزئية.

❁ السؤال هنا : ما هي الحالات التي يتحقق فيها الـ Overloading ؟؟؟ ..

- اختلاف عدد المتغيرات أو المعاملات Parameters في كل وسيلة Method.
- اختلاف نوع المتغيرات أو المعاملات Parameters ، يعني مثلا المتغير من النوع Integer يصبح String وهذا على سبيل المثال.
- اختلاف ترتيب المتغيرات أو المعاملات Parameters المررة إلى الوسيلة Method.

❁ أخشى أن تجلس وحيدا تجرب ما سبق وأن ذكرته ، وتجد بعض الأخطاء ... لذا سأوضح لك معوقات الـ Overloading :

- تغييرك لنوع إسناد المتغيرات أو المعاملات داخل الـ Method ، ونوع الإسناد هنا أقصد به الكلمات الدليلية **ByVal** وكذلك **ByRef** ، وهذا مع إبقاء المعاملات كما هي بنفس أنواعها وترتيبها .. انظر للكود التالي :

```
Public Function MySum(ByRef FirstNumber As Double, ByRef SecondNumber
As Double) As Double
        Dim x As Double
        x = FirstNumber + SecondNumber
        Return x
End Function

'-------------------------------------------'

Public Function MySum(ByVal FirstNumber As Double, ByVal SecondNumber
As Double) As Double
        Dim x As Double
        x = FirstNumber + SecondNumber
        Return x
End Function
```

ونص الخطأ يقول أن الدالتين لا يمكن تطبيق مبدأ إعادة التحميل Overloading وذلك بسبب اختلافهما في طريقة إسناد المعاملات

- يعتقد البعض أنه عند تغيير اسم المعامل Name Parameter's ، فهذا يحقق الـ Overloading ولكن لتصحيح الوضع يجب تغيير نوع المعامل أو المتغير Parameter's Type .. إليك الكود التالي ، انسخه بالمحرر وانظر إلى نص الخطأ :

```
Public Sub TestMsg(ByVal strMessage As String)
    MsgBox(strMessage)
```

```
End Sub

'------------------------------------------'

Public Sub TestMsg(ByVal intMessage As String)
    MsgBox(intMessage)
End Sub
```

- قد يعتقد البعض كحل جذري للحالة رقم (٢) والسابق ذكرها أنه إذا قام بتغيير محدد الوصول Accessibility مع بقاء ثبات الأنواع للمعاملات، فهذا قد يحقق الـ Overloading ... هل تعتقد ذلك ؟!! المفاجأة في الجواب هي (لا)، نعم لن يتحقق الـ Overloading بمجرد تغيير محدد الوصول مثل **Public** و **Private** ... الخ مع ثبات أنواع المعاملات كما هي .. انسخ الكود التالي لمحرر Visual Basic وانظر لما سيقوله لك المحرر فهو لن يقبل ذلك أبدا :

```
Public Sub TestMsg(ByVal strMessage As String)
MsgBox(strMessage)
End Sub

'------------------------------------------'

Private Sub TestMsg(ByVal intMessage As String)
MsgBox(intMessage)
End Sub
```

ولكن عليك لتحقيق الـ Overloading في هذه الحالة وقد وجب عليك تغيير محدد الوصول، يجب تغيير نوع المعاملات Parameter's Type ... فيصبح الكود بالشكل الصحيح على النحو التالي :

```
Public Sub TestMsg(ByVal strMessage As String)
MsgBox(strMessage)
End Sub

'------------------------------------------'

Private Sub TestMsg(ByVal intMessage As Integer)
MsgBox(intMessage)
End Sub
```

❁❁ ملحوظة مهمة جدا ❁❁

عند استخدامك للدوال أو ما يسمى بالـ Functions ، فيكون نفس ما تمرسنا عليه في الإجراءات مع الـ Overloading هو نفسه مع الـ Functions ، إلا أنه يوجد جزئية هامة يجب التنويه عنها.

من البديهي أن يدور بذهن أحدكم فيما يخص الـ Functions أنه إذا تم تغيير نوع الدالة أو ما يسمى بالـ Return Type فهذا يحقق الـ Overloading ... !!! ، وهنا سأصدمك بلا شك وأقول لك لن يحدث هذا ، فتحقيق الـ Overloading مع الدوال لا يأتي بتغيير نوع الدالة فقط ، ولكن لابد تحقق شرط آخر وهو اختلاف المعاملات. انسخ الكود التالي في محرر Visual Basic .NET واقرأ نص الاعتراض بنفسك :

```
Function MySum(ByVal FirstNumber As Double, ByVal SecondNumber As Double) As
Double
     Dim x As Double
     x = FirstNumber + SecondNumber
     Return x
End Function
'-------------------------------------------'
Function MySum(ByVal FirstNumber As Double, ByVal SecondNumber As Double) As
Integer
     Dim x As Integer
     x = FirstNumber + SecondNumber
     Return x
End Function
```

وكي لا أرهقك في القراءة من المحرر فنص الخطأ يفيد الآتي : " لا يمكن أن تتعدد الدالة أو لا يمكن إعادة تعريفها cannot overloaded each other وذلك لاختلافهم فقط في النوع Return Type ". أما إذا أردت تصحيح الأوضاع فإليك الكود الصحيح :

```
Function MySum(ByVal FirstNumber As Double, ByVal SecondNumber As Double) As
Double
     Dim x As Double
```

```
     x = FirstNumber + SecondNumber
     Return x
End Function
'-------------------------------------------'
Function MySum(ByVal FirstNumber As Integer, ByVal SecondNumber As Integer)
As Integer
     Dim x As Integer
     x = FirstNumber + SecondNumber
     Return x
End Function
```

❁❁ تم إضافة الدالة المسماة MySum إلى المثال OOP4 السابق ❁❁

❁❁ ملحوظة أخرى ❁❁

يستحسن أثناء تطبيق الـ Overloading ، استخدام الكلمة الدليلية **Overloads** وذلك لتسريع عمل المترجم Compiler ... فيكون كالآتي :

```
Public Overloads Sub TestMsg(ByVal strMessage As String)
MsgBox(strMessage)
End Sub

'-------------------------------------------'

Public Overloads Sub TestMsg(ByVal intMessage As Integer)
MsgBox(intMessage)
End Sub
```



خاتمة الكتاب

إلى هنا أكون قد وصلت إلى جولتي القصيرة مع OOP ، راجيا من الله - تعالى - أن أكون قد وُفقت في زيادة معرفة القارئ عن OOP ولو بشيء يسير ..

أكرر مرة أخرى ... إذا تكرمت أخي ووجدت أخطاء فنية في الأكواد أو الروابط المباشرة أو الأمثلة المرفقة أو حتى أخطاء إملائية ، فلا تتردد وراسلني كي نخرج هذا المرجع بأحسن صورة. تابعونا على منتديات أكاديمية فيجوال بيسك للعرب على الرابط www.vba4a.com أو منتديات فيجوال بيسك للعرب على الرابط www.vb4arab.com وستجدون دوما ما تطلبونه وتسعون للحصول عليه أثناء مرحلتكم التعليمية بلغات MS Visual Basic ولغات أخرى.

وانتظرونا في المرجع الشامل للبرمجة كائنية التوجه OOP في القريب بإذن الله تعالى

سبحان الله عدد خلقه ورضا نفسه وزنة عرشه ومداد كلماته ... وصلى الله وسلم وبارك
على سيد الخلق كلهم سيدنا محمد صلى الله عليه وسلم صلاة دائمة إلى يوم الدين

تحياتي



AhmedNegm@windowslive.com
002 011 977 72 44 – 002 012 944 79 49
EGYPT